Physics Analysis with the GCA Architecture

David Malon

Argonne National Laboratory

28 July 1998

STAR Collaboration Meeting
Brookhaven, NY

Aims

- **To describe how one might connect physics codes to the Grand Challenge machinery today**
- **To provide a foundation for collectively exploring how to accomplish such integration in Mock Data Challenge I and beyond**
- **To avoid detailed discussions in this forum of what the GCA architecture does, and how it works**

Physics Analysis with the GCA Architecture

If you can navigate the STAR persistent event data model:

1. Write a routine to do analysis on a single event; signature is

   void usercode(d_Ref_Any& current);
2. Link with a precompiled driver
3. Invoke the executable with an optional query string on the command line

Example

1. **Build:**

   CC -c Usercode.C

   CC -o myAnalysis Usercode.o … (GCA libraries go here)

   or

   make myAnalysis

   in the directory

   /home/common/GC/malon/BuildYourOwnGCAQuery

2. **Run:**

   myAnalysis -query “1970<num_Pion_p<1990”

   This pumps events that satisfy the query one at a time through the user’s usercode() routine.

   The user does not need to know anything about the GCA architecture or CORBA.

Sample User Code

```
#include <tagdb.h>
#include <iostream.h>

void usercode(d_Ref_Any&  inRef) {

//   PHYSICS GOES HERE.
//   This code is invoked for each event satisfying the client query.
//   This example does lambda counting ala David Zimmerman in Doug Olson's
//   oParticles HIJING event data model.

      d_Ref<oParticles> current = inRef;

      int dumbpart = current->_ptcl.size();
      int num_Lambda = 0;
      for (int i = 0; i< dumbpart; i++) {
        if( current->_ptcl[i].p.jdahep[0] == 0  ){
          if(current->_ptcl[i].p.idhep == 3122) {
            num_Lambda++;
          }
        }
      }  //end for

      cout<<"\t"<<num_Lambda<<" out of "<<dumbpart;
      cout<<" particles are lambdas"<<endl;
};
```

What if you don't want to navigate the STAR event data model directly?

Some possibilities:

- **void usercode(d_Ref_Any& inRef){**
  realUsercode(makeSTAF_Event(inRef));
  };

where makeSTAF_Event() is a library method to map a single (Objectivity) persistent event into an appropriate STAF format

or

- **wrap the iterator/driver code in STAF**

Query Builders and Drivers

- **Rudimentary query driver (show?)**
  - accepts and runs range queries
- **Less rudimentary query builder/driver**
  - accepts queries in either RangeQL (GCA) or ObjectivityQL
  - provides some query building support, error protection
  - allows users to decide whether to execute a query based on query estimates
  - enables message-level control
  - sample output (show?)

More Possibilities

No reason to require collocation of query building and physics analysis

1. **Write usercode() as above**
2. **Link with an even more elementary driver code**

   CC -o RunWithIt Usercode.o RunWithIt.o (...other libraries …)
3. **Run QueryBuilder**
   - looks like the driver code, but does NOT spawn an iterator
   - good candidate for a GUI
   - when you're ready to run, note the query token string
4. **Pass the token to RunWithIt**

   RunWithIt <queryTokenString>

More Possibilities (continued)

In less secure environments (poor man's parallelism):

```
rsh rlnx01 RunWithIt <queryTokenString>
rsh rlnx02 RunWithIt <queryTokenString>
rsh rlnx03 RunWithIt <queryTokenString>
...
rsh rlnx99 RunWithIt <queryTokenString>
```

Iterators will be fed disjoint subsets of the qualifying events

Query progress can be monitored by QueryBuilder

RunWithIt Listing

```
#include <gcaResources.h>
#include <tagdbResources.h>
void usercode(d_Ref_Any&  current);

int main (int argc, char **argv) {
  GCA_Resources gcaResources(argc, argv);
  TagdbResources tdbResources(argc, argv);
  ooInit();
  d_Transaction transaction;
  transaction.begin();
  tdbResources.openDBs(gcaResources.configuredValue("FdbBoot"));
     if (argc < 2) {
        cerr<<"You must provide a query token on the command line."<<endl;
        exit(1);
     }
     SMQ_QUERY_TOKEN_T token = argv[1];

      OrderOptIterator OOI(gcaResources.queryMonitor(), token,
                           gcaResources.messageLevel());
      while (OOI.not_done()) {
        usercode(*OOI);
        ++OOI;
      }
  transaction.commit();
  return 0;
};
```

What comprises a query?

- **A selection predicate OR a collection of event references**
- **native GCA query language (“RangeQL”) allows boolean combinations of range selections on indexed attributes**

  (1800<num_Pion_p<2000) AND (2000>num_Pion_n)

- **current TagdbResources implementation also supports ObjectivityQL queries (selection predicateson tag data members)**

  _num_Pion_zero>0.585*(_num_Pion_p+_num_Pion_n)

- **the latter uses collections-as-queries support: builds an in-memory collection from results returned by Objectivity predicatescan**

Alternative Ways to Use the Architecture Today

1. **Supply a usercode() routine, link, and run**
2. **Copy the driver code, and adapt it for personal use**
3. **Instantiate and control GCA components directly (not a good idea for production)**

Design Characteristics

- **User shielded from many ugly details**
- **GCA and CORBA are encapsulated in a GCA_Resources class**
  - encapsulates ORB and BOA references
  - encapsulates QueryFactory
  - handles configuration details, config files, and initialization
  - establishes remote connections
  - provides some utilities (message levels, access to GCA command line parameters)

This piece should need to know little, if anything, about Objectivity.

Design Characteristics II

- **Tagdb and Objectivity details are encapsulated in a TagdbResources class**
  - federation and tagdb handle management, opens and closes
  - query builder utilities
  - tag-content-specific stuff
  - support for ObjectivityQL Tagdb scans

  This piece should need to know little, if anything, about the GCA architecture and CORBA.
- **Currently the locus for Objectivity-to-CORBA utilities**
- **In a mixed environment, coordination will be needed for session control.**

Saving query results

- **What to save:**
  - OIDs of events that satisfy a selection predicate
  - OIDs of events that survive a usercode() cut
  - more than just OIDs?
- **GCA has begun exploring rudimentary approaches to the first two of these**
  - in Objectivity
  - in external files
- **support of third option would require more than GCA tools**
- **saving in Objectivity requires resolution of access control issues**

“Standard” GCA drivers open the federation in read-only mode.